The Virtual Life of ENIAC
Simulating the Operation of the First Electronic Computer

Till Zoppke and Raul Rojas
Dept. of Mathematics and Computer Science
Freie Universität Berlin
Takustr. 9
14195 Berlin, Germany

Abstract

This paper describes an interactive simulation of the Electronical Numerical Integrator and Computer (ENIAC). With the simulator, the user can wire up complex configurations of the ENIAC modules. The simulation can be started in continuous run modus, or can be advanced step by step. The simulation has been written in Java and can be started from an Internet site. The simulation has been run using a datawall six meters long which provides the closest avalibale approximation to the look-and-feel of programming this historical computer.

1 Introduction

Sixty years ago, in February 2006, the ENIAC (Electronic Numerical Integrator and Computer) was officially presented to the public at the Moore School of Engineering of the University of Pennsylvania [1]. The story of the ENIAC has been told many times, but beginning with the 50th Anniversary in 1996, there has been a new surge of interest about the machine and its functionality. Jan van der Spiegel and his team of students at UPenn built a functional replica of the main modules of the ENIAC using a single chip [2]. *ENIAC-On-A-Chip* is a fitting tribute to the machine which started the era of electronic computers and is also materialized history of technology, living proof that we still know, many decades later, how the ENIAC worked in detail.

This contribution describes a software simulation of the ENIAC developed by the authors at Freie Universität Berlin in Germany. The simulation can be started from any computer connected to the Internet. With the simulation, any interested person can program the ENIAC to solve numerical problems. "Programming" means here wiring the machine, that is, laying down connectors between the functional units so that information can flow synchronously from one module to the next. Expressed in modern terminology, ENIAC was a *parallel dataflow machine*. Our simulation lets the user fully appreciate this fact. The user interface allows her to interconnect modules using drag-and-drop virtual cables. Once the machine has been wired, it is possible to start the computation, follow the operations step by step, and make the machine go alternatively faster or more slowly.

A number of simulations of historical computers exist. A string of simulators for the EDSAC, the first computer built at Cambridge University in the late 1940s, have been written since 1978 [3]. The EDSAC simulator allows the user to see the contents of the cathode ray tube memory, modify it, and run programs using a graphical interface. A simulation in Java of Konrad Zuse's Z3 machine is also available on the web [4]. The software shows the internal state of the machine and lets the user write programs using a punched tape. This was probably the first Java simulator of a historical computer which became available on the Internet. There

exist quite a few other simulators and emulators for many more recent computers: for Digital Equipment's PDP-11, for the PDP-8, for the Commodore 64, and even old mainframes.

Why do people write simulators for historical machines? In some cases it is pure nostalgia: most programmers have fond memories of the computers they learned to program with. The machine has to be preserved in virtual form when the hardware becomes obsolete. In our case, writing simulations of historical machines is something that allows us to get computer science students interested in the history of the subject. Engineers are artisans: they best learn the history of technology by recreating the past with their own hands. The result is educational, but also useful for historians of technology and the public at large. Martin Campbell-Kelly wrote about the EDSAC simulator: "Thus the primary aim of the EDSAC simulator has not been so much to create an "EDSAC experience" as to afford a vehicle for the scholarly study of an early programming system by today's university students and computer professionals. As it happens, many users have reported a real sense of walking in the shoes of the original EDSAC programmers" [3].

Our simulator of the ENIAC allows the user to "walk in the shoes" of the ENIAC programmers, the legendary "ENIAC girls". The user interface and the wiring of the simulated machine is as faithful to the original as possible. The complexities of ENIAC's programming and timing can be best understood by making the same errors as were probably made in the 1940s by the pioneers. However, a software simulation is more flexible than real hardware: errors are never catastrophic, debugging is easier, successful machine configurations can be stored and loaded on demand. Although we have not implemented all the modules of the ENIAC, we have a working simulation of the accumulators, constant units, initiating unit, and others more, which allow the programmer to assemble fairly complex configurations of the machine as described further down.

This article is organized as follows: First we give a bird's eye overview of the ENIAC architecture, and then we describe our simulator. A worked-out example of ENIAC wiring makes easier to understand how the machine was programmed. We finish the article with some conclusions and references to future work.

2 The ENIAC Architecture

Fig. 1 shows a diagram by van der Spiegel of the abstract architecture of the ENIAC, with its five functional parts: a) arithmetic, b) I/O, c) global control and programming unit, d) memory for tables, and e) busses.

Each of the 20 accumulators can store a positive or negative decimal number (with up to 10 digits, in tens-complement representation). A number arriving into an accumulator is always added to its previous contents – hence the name accumulator. Subtraction is achieved by transmitting the tens complement of a number to an accumulator (that is, instead of transmitting a number a we transmit $-a$). There is a single multiplication unit, as well as a single divider & square root unit. Constants are entered into the machine using decimal dials mounted in the constant transmitter panels, or reading IBM punched cards. A printer can be used as output device for numerical results. Therefore, not considering the function table units, nor the constants which can be set up with dials, the ENIAC operated with only 20 read/write memory cells.

The cycling unit provides the timing for the whole machine by generating and distributing ten different trains of pulses. The other units can use these pulses as reference or as input, so that the whole machine remains synchronized. It is the equivalent of today's clocking circuits in computers. The initiating unit, on the other hand, allows the operator to start or stop the machine, and also to reset the circuits at the beginning. The master programmer unit can be used to implement loops, or to start two different computations in parallel in two portions of the machine.

There are two busses in the ENIAC: the control bus is used to send input pulses to start the units. A unit which finishes a computation sends an output pulse, through the control bus, to the next unit, in order to start the next computation. The data bus is used to transfer decimal numbers from one unit to another. Each decimal digit is transmitted through a cable line using from zero to 9 pulses, according to the digit. Ten lines in a connector cable can transmit ten digits in parallel, that is, the contents of an accumulator. The units are not permanently connected to the busses: it is precisely the job of the programmer to lay down the appropriate connections from each unit to the next, using cables, thus effectively laying down the busses through which information flows. In the ENIAC the busses were just trays for holding the cables connecting the units together.

For a more detailed discussion of the architecture of the ENIAC see [2], [5], [6], [7], and [10] in this issue.

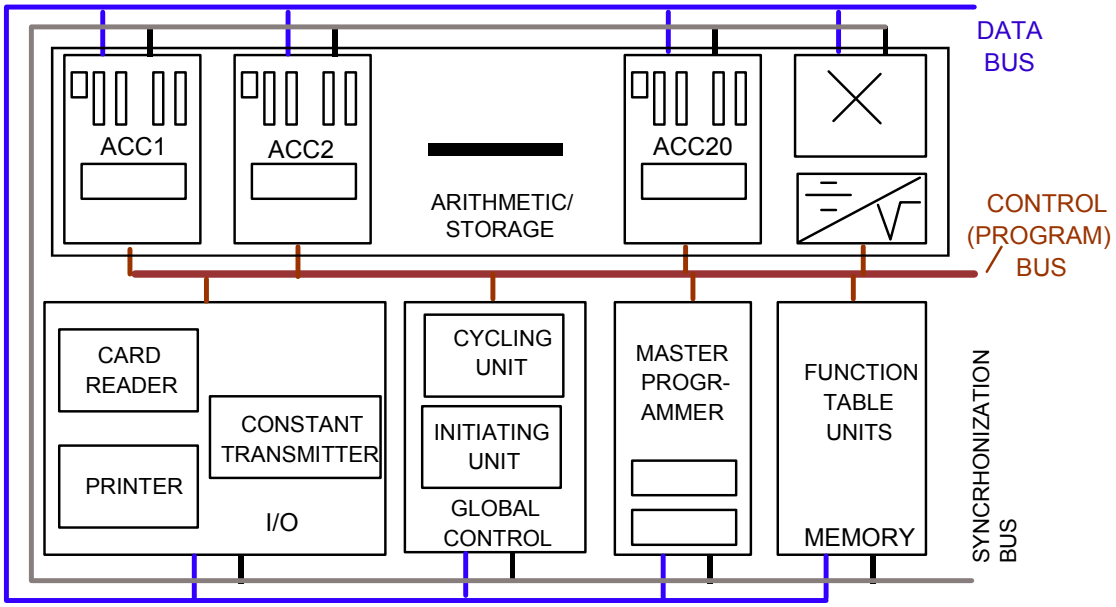

Figure 1: Diagram of the ENIAC functional units. Reproduced from [2].

Fig. 2 shows an abstract diagram of the accumulators, which allows us to understand its wiring. One can think of an accumulator as an addition box containing a single ten digit number. There are five possible numerical input connections (labeled α, β, γ, δ, and ε). There are two possible output connections (labeled A and S). An input pulse and its corresponding settings (in the boxes near to the input pulse) tell us how the accumulator is used. The first example in Fig. 2 is that of an accumulator where the α input connection has been selected, and is used seven times. That is, this accumulator will receive a ten-digit number through connector α seven times. At the end of the seven additions, an output control pulse is generated to signal completion of the task. In the second example in Fig. 2, the accumulator

accepts three times a ten-digit number through connector γ. In the third example, the accumulator outputs its contents through connector A twice. The selection of connector, therefore, determines if the accumulator consumes an input or generates an output number.

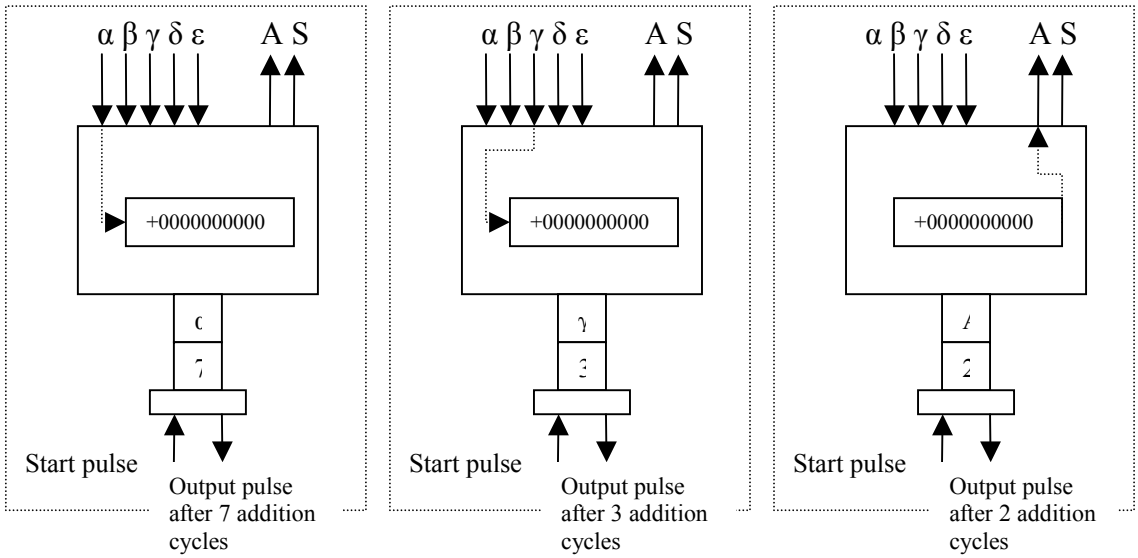

Figure 2: Diagram of an accumulator showing three connection examples. See main text for the explanation.

If the dials in an accumulator could be used only once (as in the examples in Fig. 2), then 20 accumulators would not be enough for complex and intricate calculations. What the ENIAC engineers did, was to put 12 connectors for input pulses in each accumulator, with their corresponding α-ε and repetition switches. Each unit can then be used up to twelve times in the dataflow diagram of a computation. Theoretically this gives us 240 components we can use in the computational dataflow. Since many computations are performed only once and do not need to start another unit, the first four input connectors were built so that they do not generate an output pulse and execute its operation only once.

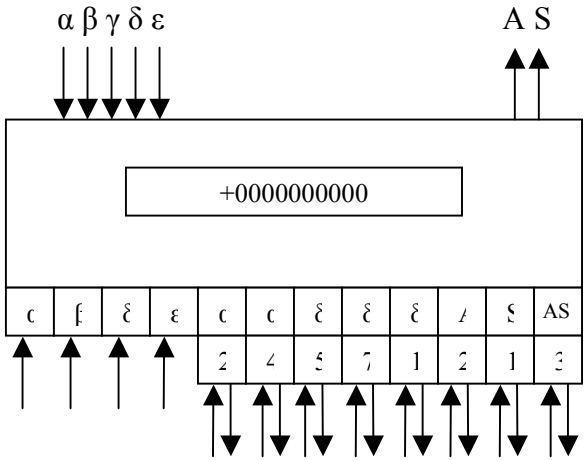

Figure 3: An accumulator has twelve connectors for input pulses. The first four do not generate an output pulse at completion, the last eight do. Each input pulse has a dial to select the connector through which a number will arrive (α, β, γ, δ, ε) or will leave the machine (A, S, or both). The first four input pulse connectors lead to a single repetition of the operation. The last eight connectors can repeat the operation from 1 to 9 times.

Fig. 3 shows a more complete abstract diagram of an accumulator. It can only hold a single ten-digit number, but it can be started in twelve different ways, as shown in Fig. 3: reading from the α, β, δ, and ε connectors, and executing once (first four input pulse connectors), or reading from the α, α, δ, δ, δ connectors 2, 4, 5, 7 and 1 times, respectively. It can output its contents through A or S, twice or once, or it can output both the contents and its ten-complements through A and S simultaneously, three times (last connector). Accepting a number through the same connector in two different cases (say connector α) is not a contradiction -- the unit sending the number through the bus and α can be started also in two different ways so that it sends the number 2 or 4 times.

One example, taken from [2], will help us to see how accumulators can be used in a complex calculation. In Fig. 4, the accumulators 4, 5 and 6 are used to compute (*a*-*b*), (*b*+359), and (*c*+*2b*+359) in parallel, where *a*, *b*, and *c* are the initial contents of the accumulators 4, 5, and 6. The cables A1, A2, I, II, and III have been laid out by the programmers. The start pulse arrives through cable A-1 and triggers the three accumulators concurrently. Accumulator 4 receives the number sent from accumulator 5, from the S (subtraction) connector, through cable I. Accumulator 4 thus gets (-*b*) through the input channel α. Therefore, at the end of the first computation cycle, accumulator 4 will contain (*a*-*b*).

Accumulator 5 switches are positioned so that it outputs its contents twice. The "A" output connector is directly routed to accumulator 6 using cable II. The switches in accumulator 6 are positioned so that it reads from connector α twice, so that at the end of the second computation cycle, the content of accumulator 6 is the number (*c*+2*b*). Next, accumulator 5 sends a "finished" pulse at the end of the second computation cycle through connector A2. This pulse triggers accumulators 5, 6, and the constant transmitter. This last unit sends the constant 359, set by hand by the programmers using decimal dials, into both accumulator 5 and 6 through cable III. Accumulator 5 receives this number in input connector α, and accumulator 6 in connector β. At the end of the third and last computation cycle, the contents of the accumulators are, respectively, (*a*-*b*), (*b*+359), (*c*+2*b*+359). For a more detailed description of ENIAC programming see [6] and [7].

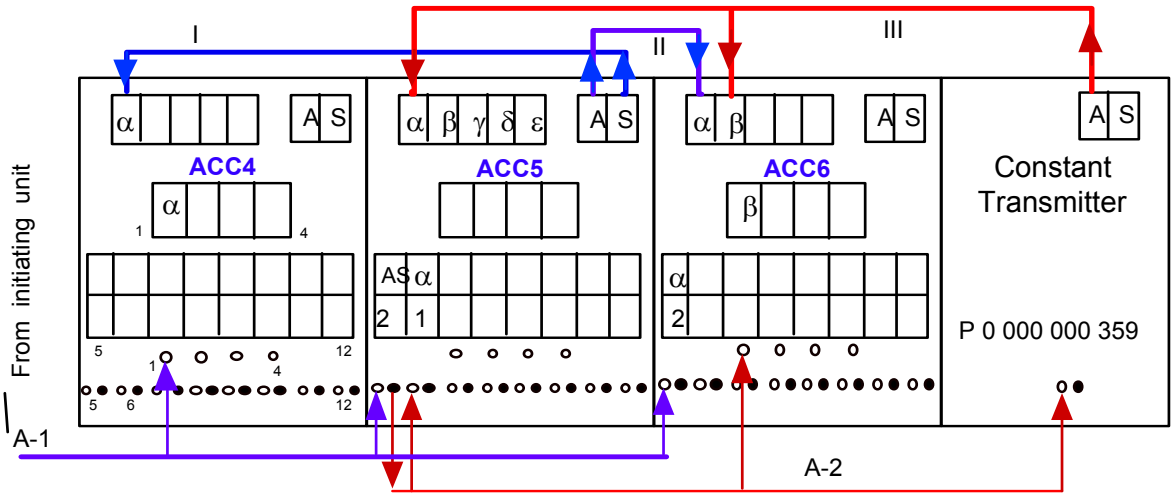

Figure 4: The connections for an ENIAC program. Three results are computed in parallel in each accumulator. Taken from [2].

As this simple example shows, the main task for the programmer is figuring out where to store partial results, and how to continue generating new results. All accumulators can

compute one addition in parallel. A unit which has finished can start one or more units, even itself again, through output pulse connectors. A unit can be started with a pulse going into any of its 12 possible input connectors. A unit which has been started can repeat its computation from 1 to 9 times, according to the position of a switch.

3 The Simulation

The simulation is started in a Java capable browser (JRE 1.4+), from the hard disk or from the Internet. On start up, the user is prompted to load a machine configuration. Both blank configurations in different sizes (from 2 up to 16 accumulators) and complete programs (e.g. Fibonacci number computation) are available. Alternatively the user can load a configuration developed and stored in a previous session.

When the configuration has been loaded, the ENIAC programmer sees a window with all units ordered side by side. So, depending on the number of units to be shown, the window might be quite wide. So for a better navigation an overview window is optional. In Fig. 5, two accumulators, the Initiating unit and the Cycling unit are visible, while the complete configuration consists of twelve units. The graphical representation is clearly not a photo-realistic reconstruction of the ENIAC, it is more of a stylized representation. However, in our program it is possible to substitute the "skin" of each panel with any given graphical representation, that is, it would be possible to use a photograph of the ENIAC panels. This could be done in the future, to increase the realism of the simulation. In the Initiating unit we pasted a photograph of an operator in front of the control panels of ENIAC, as if you were looking on a mirror.

The simulation is interactive. Switches and buttons can be manipulated with the mouse. A cable connection can be laid out by clicking on any connector and dragging the cable (which appears under the cursor) to another connector, releasing then the mouse button. The simulation is fairly intuitive for anybody familiar with computer GUIs. The biggest obstacle though, is not the user interface itself but understanding how the ENIAC works in order to wire the machine in a sensible way.

Once all cabling is complete, the user turns on each panel using its on-off switch. Then he presses the "go" button of the Initiating unit, delivering the initial pulse in order to start crunching numbers. The operation of the machine can be observed: Whenever an electrical pulse is sent through a cable, the cable is highlighted. Intermediate results appear at the lamps of the accumulators. Finally, when the data flow dries up, the result can be read out from the accumulator lamps.

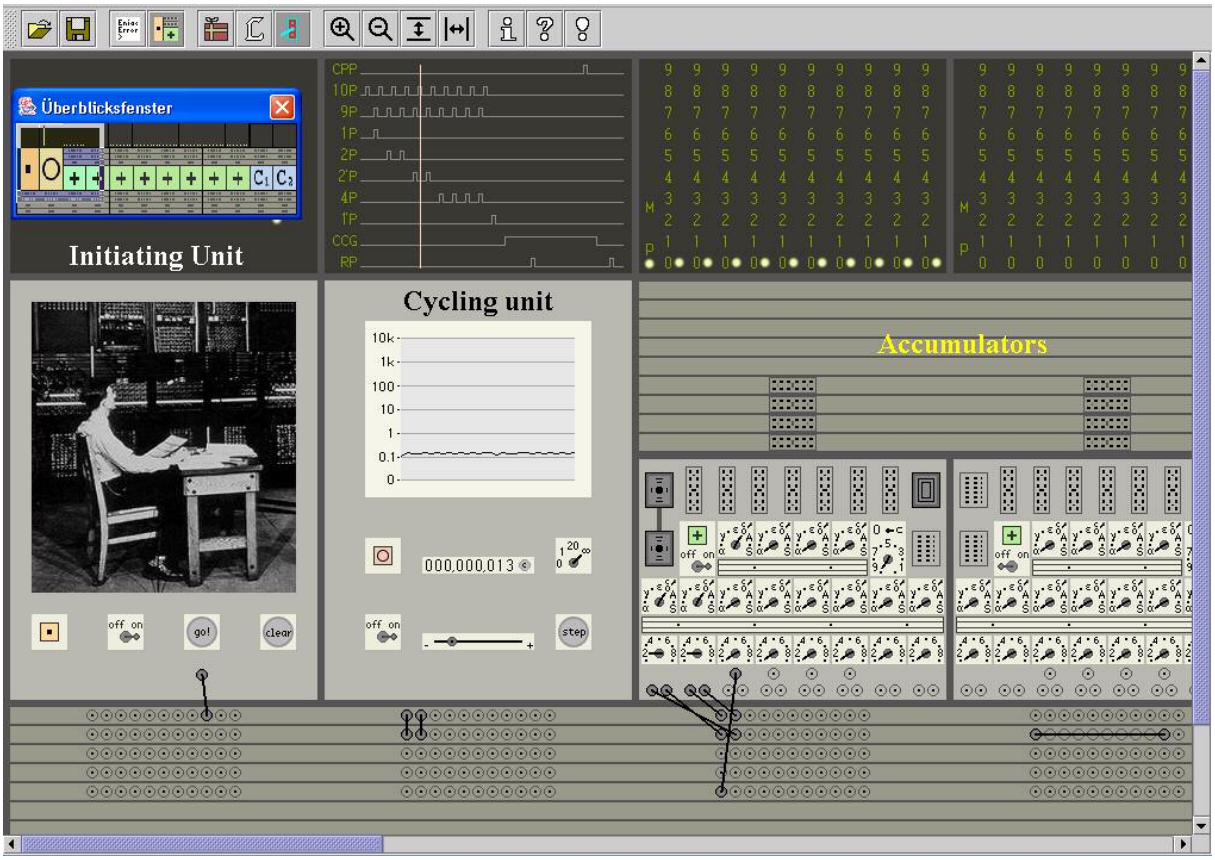

Figure 5: Screenshot of the Initiating Unit, Cycling Unit, and two accumulators (labeled). The first accumulator is switched on and shows its contents (ten decimal digits, all zero). The second accumulator is switched off. A few cables have been laid down, interconnecting the units to the control bus.

Let us review the components of the simulation one by one, in order to further clarify the way the ENIAC was used. Look first at the screenshot of the Cycling Unit in Fig. 6. The upper panel shows the ten trains of pulses generated by the unit, while the moving cursor indicates the current state of the trains (that is, the point in time within the 20-cycles computation period). The clock of the machine can be controlled by the slider, and the effective speed is displayed in the simulated oscilloscope in kHz (the historic ENIAC's clock operated at 5 kHz). The switch in the middle right can be set so that the Initiating unit advances 1/3 of a cycle, 1 cycle, 20 cycles (20 cycles is the time it takes to perform 1 addition in an accumulator), or keeps on generating pulses infinitely. When the switch is set to a non-infinite value and the machine stops, it can be reactivated for another 1/3, 1 or 20 cycles by pressing the "step" button. This is the easiest way to debug a program. The counter in the middle of the unit marks the total number of addition cycles from the start of the computation.

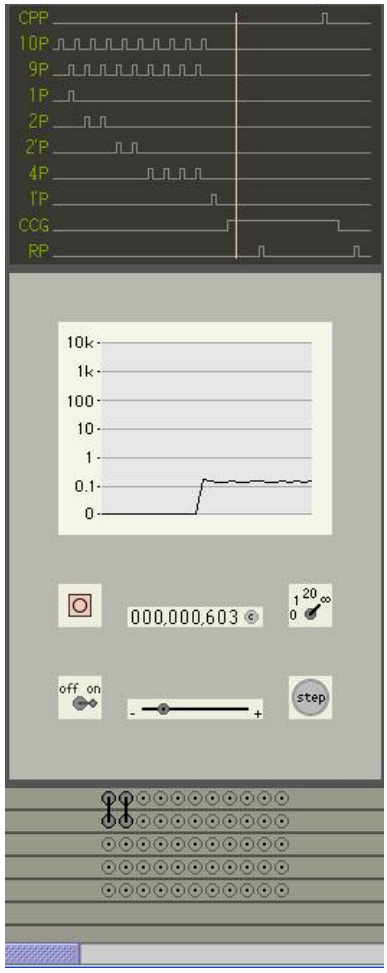

Figure 6: Detail of the Cycling Unit in the simulation. For an explanation see the main text.

In comparison, the Initiating Unit is rather simple. The "clear" button is used to reset all accumulators and circuits. "Go" is the button to start the computation mentioned before. There were further elements of the historic Initiating unit, used for controlling the ENIAC's booting process. For simplicity, some of them have been omitted—they are pointless for the operational simulation because they have to do with the initialization of vacuum tubes.

An accumulator has a more intricate structure. It can store a number, which is displayed on the decimal panel on top of the controls using lamps (Fig. 7). Computation can be triggered using the twelve different connectors mentioned above through which it can receive an initial pulse. In Fig. 8, the connectors are visible below the switches, and have been labeled "I" (for input pulse) and "O", for output pulse. Eight of the input connectors have to the right an output connector, with which another unit can be started at the end of the execution cycle. The four input connectors without a corresponding output connector can trigger an operation only once. The other eight can repeat the same operation up to nine times. That is, if, for example, accumulator 4 reads five times from accumulator 8, then accumulator 8 is set to transmit five times its contents and accumulator 4 is set to receive it also five times. Both accumulators have to be started simultaneously.

In Fig. 8 (lower diagram), some of the details have been erased so that the correspondence between switches and connectors becomes clear. The encircled components in Fig. 8 show the input connector which starts an operation and the corresponding switch which determines which decimal port will be used to read or write a number (α, β, γ, δ, ε, A, S, or AS). The

encircled switch with a value between 1 and 9 determines the number of repetitions of the operation.

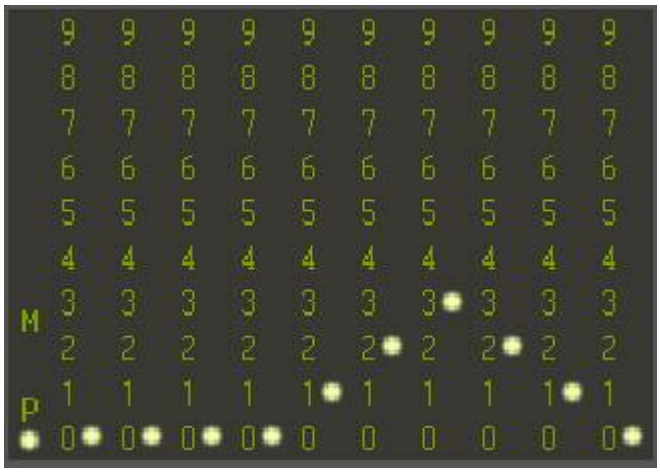

Figure 7: The lamps in the accumulator show its contents, in this case the number +0000123210

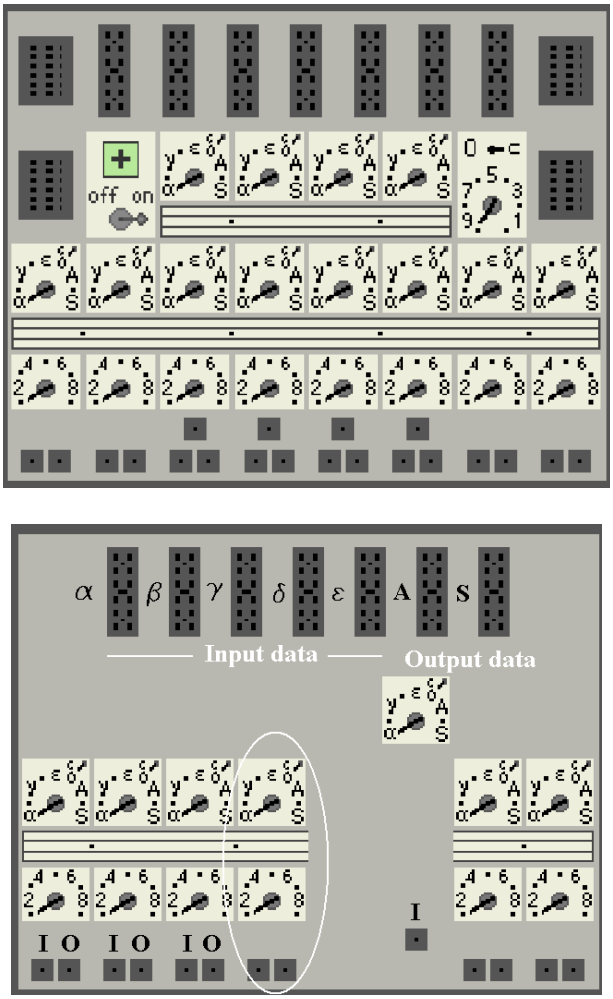

Figure 8: On top, the switches in an accumulator. Below, some components have been deleted in order to show the correspondence between switches and connectors. See main text.

Fig. 9 shows a screenshot of two accumulators cabled so that the contents of the accumulator to the right is subtracted from the accumulator to the left. In the lower image in Fig. 9, all

controls which are not being used for this special operation have been removed from the graphics, so that only the essential connections are visible.

The initiating pulse arrives through the rightmost connector in the highest trunk of control cables. The accumulator to the left is started, and the corresponding switch for the input connector selected is set to “alpha”. That is, a number will be received as input through that connector. The accumulator to the right is started at the same time -- it is set to output the tens complement of the accumulator contents through the connector “S” once. If the contents of the accumulators are *a* and *b*, at the end of the operation the results (*a*-*b*) respectively *b*, are stored in the two accumulators.

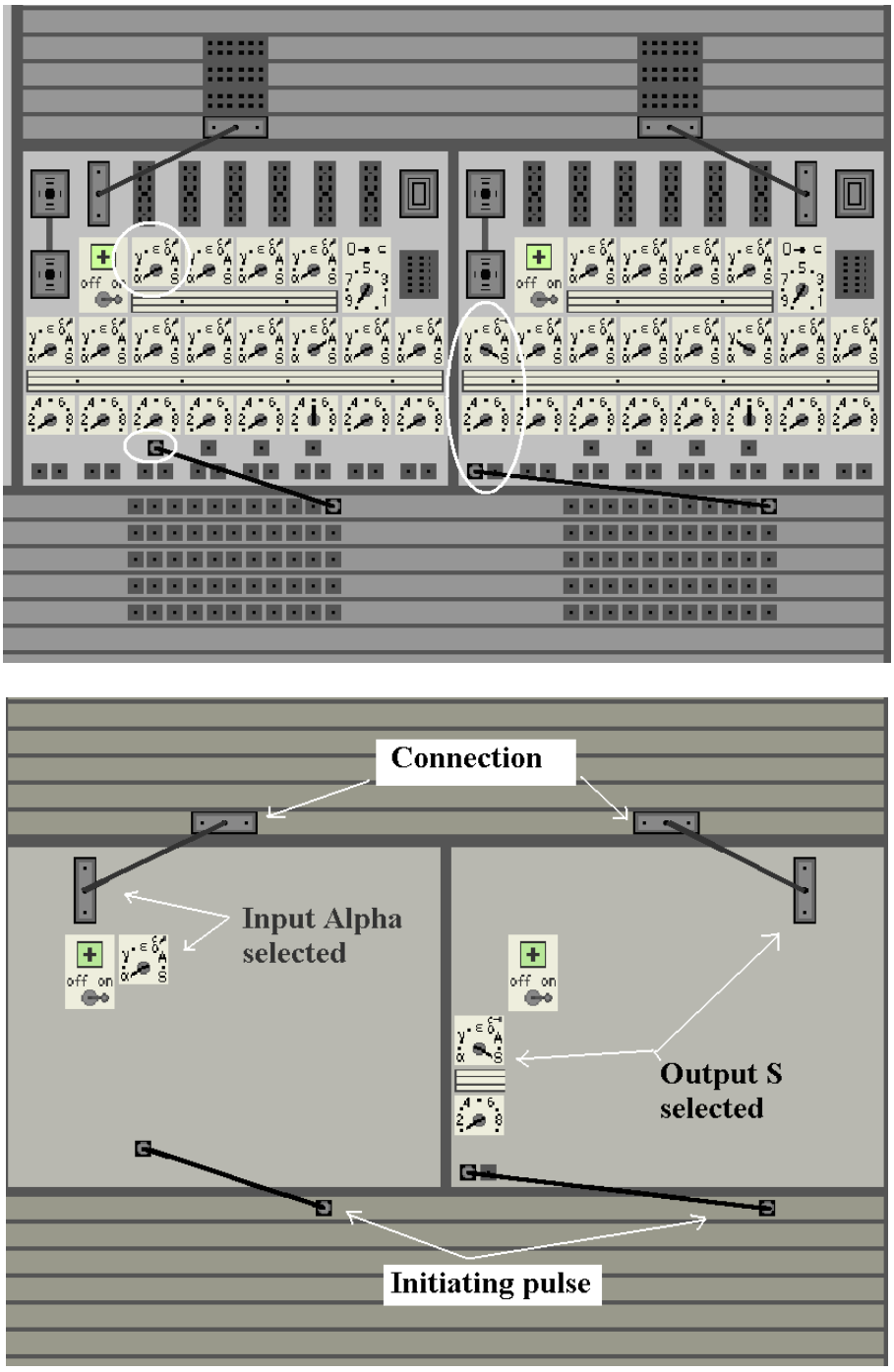

Figure 9: Two accumulators connected to subtract the contents of the accumulator to the right from the accumulator to the left. Only the relevant components are shown in the lower image.

There are some other details which have to be considered in order to finish wiring the machine. The online documentation of the simulation provides all the necessary information.

4 A simulation example

Fig. 10 shows the machine configured to run the program whose cabling is shown in Fig. 2. In order to simplify this example, the constant transmitter was substituted with an accumulator. The accumulators contain the numbers 1, 2, 3, and 5, respectively. The computation performed by this arrangement is the same as the one discussed in section 2.

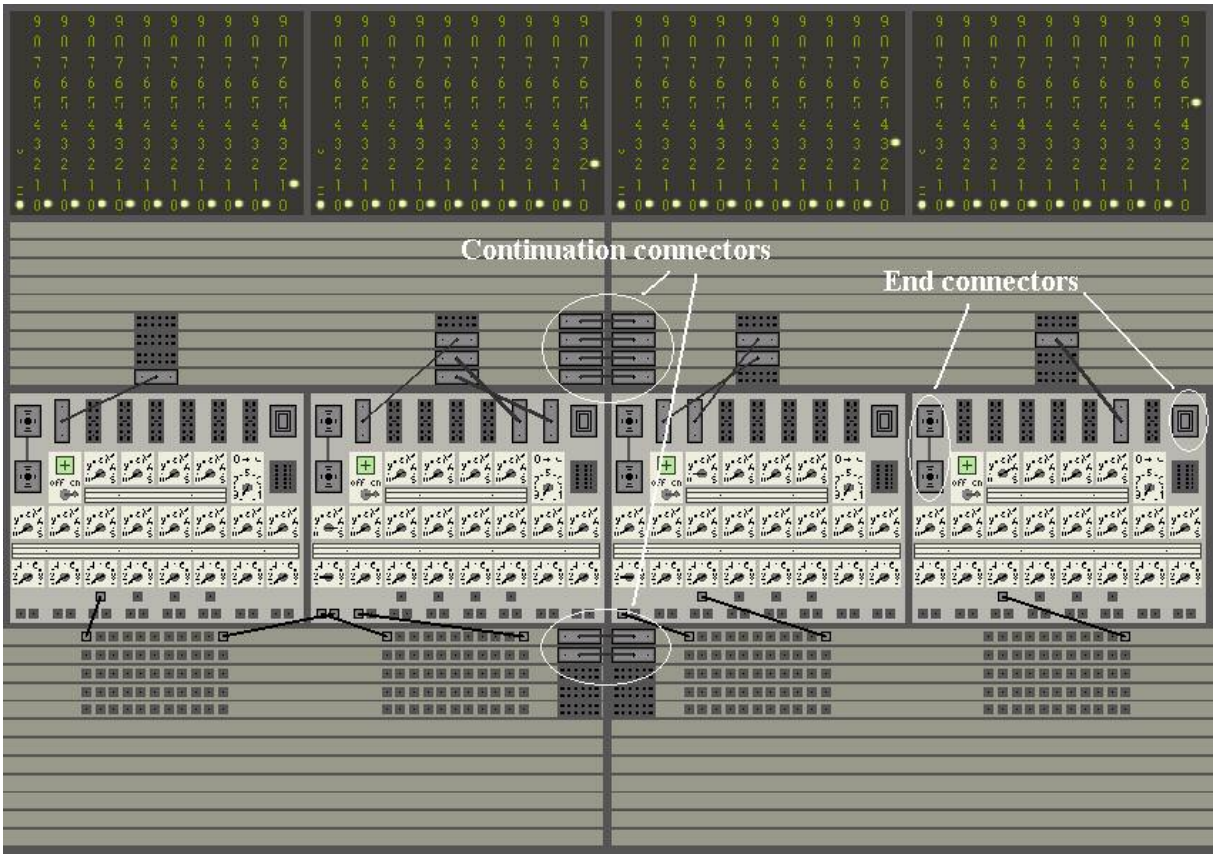

Figure 10: Cabling of the ENIAC simulation corresponding to the example of Fig. 2

The white ovals in Fig. 10 enclose some connectors necessary because of the way the trunks and accumulators were built. The trunks contained cables running below or above the control panels of up to four units in a row. To continue the signals, a connector cable is placed between the end and the begin connector of two consecutive trunk segments. It is just a way to make the data and control busses longer. Also, two accumulators can be interconnected to perform a 20-decimal digit computation. If the accumulators are used for 10-digit numbers, then the end connectors are placed as shown in the figure.

This configuration is started by connecting the Initiating unit to the first control pulse connector in the control trunk, setting all units on power, and pressing the “go” button. During the computation, the contents of the accumulators changes as in the original ENIAC. When a number is given off, all accumulator digits go through a complete rotation through the lamp field. This is the most colorful part of the simulation.

5 Conclusions and future work

In this paper we have discussed how the ENIAC simulation is used. Details about the software can be found in [9], as well as a tutorial on programming the ENIAC. The simulation is not yet complete, it will continue to grow in the next years and will hopefully cover all modules of the machine. A first version of the simulator, without the graphical interface discussed here, was written by Peter Hansen under the supervision of R. Rojas [8]. The new version, presented here, is more complete and immersive.

We hope that this paper can convey to the reader a feeling of how the ENIAC was programmed. We hope Annals readers decide to test-drive the simulation and find out by himself or herself how this machine worked. ENIAC was a marvel of engineering, even if its inventors soon realized, already during its construction, that much could be improved or simplified. It was a massive machine, filling a whole room. A further approximation to the experience of programming the ENIAC can be obtained by letting the simulation run in a multiscreen, rear-projection system, as shown in Fig. 11, where we see the simulation running in four screens. The screens are touch sensitive and the user can configure the machine walking from one module to the next. There are plans to let the simulation run using even more rear projectors during "ENIAC NOMOI", an art performance being prepared in Berlin for 2006. A user standing in front of the machine can literally "walk in the shoes" of the first programmers. Such multiscreen systems could become standard in computer history museums in the future, when OLED or other kinds of cheap large plastic computer displays become available.

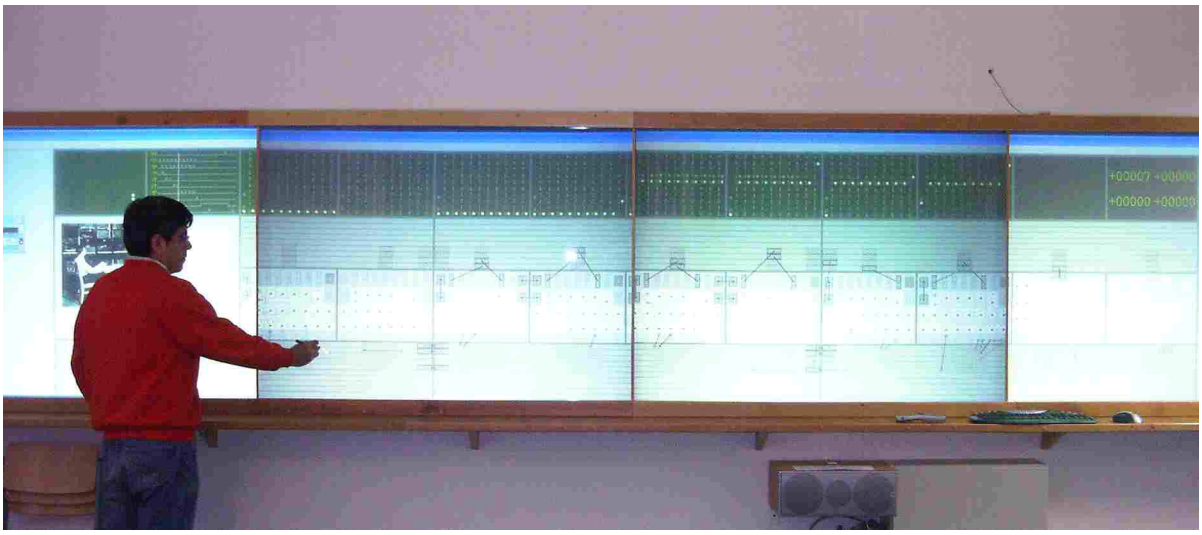

Figure 11: The ENIAC simulation running in the touch sensitive data wall installed in the Department of Mathematics and Computer Science, Freie Universität Berlin. The wall is six meters long and can display up to eight ENIAC units simultaneously.

In 1998, Jan van der Spiegel wrote about ENIAC-On-A-Chip: "The project gave the design team a first-hand appreciation and understanding of the workings of the ENIAC, from the architectural level down to the functional, programming, and circuit levels" [2]. The simulation of the ENIAC has the same purpose: to motivate a new generation of young students to understand first-hand a historical machine, by working and playing with it, and to preserve the history of computing by giving ENIAC a new, a virtual life.

The simulation is available online at www.zib.de/zuse.

References

[1] S. McCartney, ENIAC: The Triumphs and Tragedies of the World's First Computer, Berkley Publishing Group, 2001.

[2] J. Van der Spiegel, J. F. Tau, T. F. Ala'ilima, and L. P. Ang (2000). The ENIAC: History, Operation and Reconstruction in VLSI. In: R. Rojas, U Hashagen (eds.), *The First Computers - History and Architectures*, MIT Press, Cambridge, 2000.

[3] M. Campbell-Kelly, "Past into Present: The EDSAC Simulator", In: R. Rojas, U Hashagen (eds.), *The First Computers - History and Architectures*, MIT Press, Cambridge, 2000.

[4] A. Thurm, „Eine Simulation der Z3 für das Internet", in R. Rojas, *Die Rechenmaschinen von Konrad Zuse*, Springer-Verlag, Berlin, 1998.

[5] A Report on the ENIAC (Electronic Numerical Integrator and Computer), Contract No. W-670-ORD-4926 between Ordnance Department, United States Army Washington, D.C. and University of Pennsylvania, Moore School of Electrical Engineering, Philadelphia, PA, June 1, 1946 (available online at http://ftp.arl.mil/~mike/comphist/46eniac-report/index.html).

[6] H. H. Goldstine, A. Goldstine, "The Electronic Numerical Integrator and Computer (ENIAC)", 1946, reprinted in B. Randell (ed.), *The Origins of Digital Computers: Selected Papers*, Springer-Verlag, New York, 1982, pp. 359-373.

[7] Arthur W. Burks, Alice R. Burks, "The ENIAC: The First General-Purpose Electronic Computer", in *Annals of the History of Computing*, Vol. 3 (No. 4), 1981, pp. 310-389.

[8] P. Hansen, „A Java Simulation of the ENIAC", Bachelor Thesis, Computer Science Department, University of Osnabrück, unpublished.

[9] T. Zoppke, „Simulating the ENIAC as a Java Applet", Diploma Thesis, Computer Science Department, Freie Universität Berlin, June 2004, unpublished.

[10] H. Neukom, "The Second Life of ENIAC", in this issue.